Type inference with constraints

Jose Vergara

University of Technology Sydney

Sydney Area Programming Languages INterest Group 2010

Types and type Systems

A type is a formal description of the behavior of a program fragment. For example:

- $plus \ : Int \rightarrow Int \rightarrow Int$
- $map : (a \rightarrow b) \rightarrow List\ a \rightarrow List\ b$
- $select : (all\ a.a\ \rightarrow\ List\ b)\ \rightarrow\ c\ \rightarrow\ List\ b$

The Damas-Milner type system [Milner, 1978] offers a restricted form of polymorphism and is at the heart of Standard ML and Objective Caml.

The Dammas-Milner type system us capable to type functions like *plus* and *map*, but what about the *select* function?

```
let ext (select: (all a . a -> List b) -> c -> List b) =
fun f ->
| z y -> append (f (z y)) (append (select f z) (select f y))
| y -> f y
;;
```

It is necessary to have:

- Local quantification and type application
- Type matching for the function of type $\forall a.a \rightarrow List\ b$

System FM

System FM extends system F with type matching.

$$\begin{array}{lll} t & ::= & & (term) \\ & & x^T & (variable) \\ & & t\ t & (application) \\ & & \lambda x^T.t & (abstraction) \\ & & t\ T & (type\ application) \\ & & [\Delta]T \to t & (typecase) \end{array} \qquad \begin{array}{lll} T & ::= & & (type) \\ & & X & (variable) \\ & & T \to T & function) \\ & & \forall_T[\Delta].T & (typecase) \end{array}$$

Let Δ be a type context and let P and U be types. There is a match of P against U with respect to $\Delta\ iff\ \{U/[\Delta]\ P\}$ is some substitution in which case is their most general (Jay; 2009).

Type unification

Type unification for system FM is simple:

$$\begin{array}{lcl}
\{X = X\} & = & \{\} \\
\{S = X\} & = & \{S/X\} if\ X \notin FV(S) \\
\{X = T\} & = & \{T/X\}\ if\ X \notin FV(T) \\
\{P \to S = Q \to T\} & = & let\ \upsilon_1 = \{P = Q\}\ in \\
 & & let\ \upsilon_2 = \{\upsilon_1 S = \upsilon_2 T\}\ in \\
 & & \upsilon_2 \circ \upsilon_1 \\
\{\forall_P[\Delta].S = \forall_Q[\Delta].\} & = & let\ \upsilon_1 = \{P = Q\}\ in \\
 & & let\ \upsilon_2 = \{\upsilon_1 S = \upsilon_2 T\}\ in \\
 & & \upsilon_2 \circ \upsilon_1\ if\ this\ avoids\ \Delta \\
 & & \\
\{S = T\} & = & undefined\ otherwise
\end{array}$$

Full type inference for System F is undecidable (Wells; 1999). In order to support first class polymorphism **bondi** relies on provided type annotations to correctly type functions with polymorphic type arguments.

It is also desirable to have an efficient type inference algorithm like those of the OCaml and Haskell compilers.

So far we have a specification of the problem, it remains to:

- Provide the proofs for type unification(in progress) and type matching.
- Define and implement type unification with constraints.
- Deal with type annotations when performing type inference with constraints on path polymorphic queries.

What to do?
As a first step, lets learn how to do type inference with constraints to support the Dammas-Milner type system.

Typing rules for DM

$$t ::= x \mid t\ t \mid \lambda x.t \mid let\ x\ =\ t_1\ in\ t_2$$
$$T ::= X \mid T \rightarrow T \mid G$$
$$\sigma ::= \forall \bar{X}.T$$

$$\frac{\Gamma(x) = S}{\Gamma \vdash x : S}\ (Var) \qquad \frac{\Gamma \vdash t_1 : U \quad \Gamma; x : U \vdash t_2 : S}{let\ x = t_1\ in\ t_2 : S}\ (Let)$$

$$\frac{\Gamma; x : U \vdash t : S}{\Gamma \vdash \lambda x.t : U \rightarrow S}\ (Abs) \qquad \frac{\Gamma \vdash t : T}{\Gamma \vdash t : \forall X.T}\ X \notin ftv(\Gamma)_{(Gen)}$$

$$\frac{\Gamma \vdash t_1 : U \rightarrow S \quad \Gamma \vdash t_2 : U}{\Gamma \vdash t_1\ t_2 : S}\ (App) \qquad \frac{\Gamma \vdash t : \forall X.T}{\Gamma \vdash t : [X \mapsto U]T}\ (Inst)$$

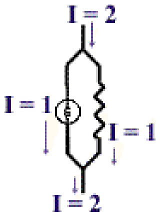
Algorithm W

$$W \;::\; TypeEnv \times Expression \;\rightarrow\; Substitution \times Type$$

$$\begin{array}{lcl}
W(\Gamma, x) & = & ([\,], instantiate(\tau)), where (x : \tau) \in \Gamma \\
W(\Gamma, \lambda x \rightarrow e) & = & let (\sigma_1, \tau_1) = W(\Gamma \backslash x \cup \{x : \beta\}, e), fresh \beta \\
 & & in\ (\sigma_1, \sigma_1 \beta \rightarrow \tau_1) \\
W(\Gamma, e_1\ e_2) & = & let\ (\sigma_1, \tau_1) = W(\Gamma, e_1) \\
 & & in\ \ (\sigma_2, \tau_2) = W(S_1 \Gamma, e_2) \\
 & & \sigma_3 = mgu(\sigma_2 \tau_1, \tau_2 \rightarrow \beta), fresh \beta \\
 & & in\ (\sigma_3 \circ \sigma_2 \circ \sigma_1, \sigma_3 \beta) \\
W(\Gamma, let\ x\ =\ e_1\ in\ e_2) & = & let\ (\sigma_1, \tau_1) =\ W(\Gamma, e_1) \\
 & & in\ (\sigma_2, \tau_2)\ =\ W(\sigma_1 \Gamma / x \cup \{x : generalize(\sigma_1 \Gamma, \tau_1)\}, e_2) \\
 & & in (\sigma_2 \circ \sigma_1, \tau_2)
\end{array}$$

Constraints

A constraint is a condition that a solution to an optimization problem must satisfy. A simple example of a constraint problem is: *At any junction point in an electric circuit, the total electric current into the junction is equal to the total electric current out.*



Wikipedia (2010).

Defining constraints for the DM type system

Algorithm W work with substitutions that are an approximation to solved forms of unification constraints. Working with constraints means using equations, conjunction and existential quantification (Pottier and Rémy; 2005).

A type is either a type variable X or an arity-consistent application of a type constructor F. The type constructors are: $unit$, $\times$, $+$,$\rightarrow$, etc...)

Ground types contain no variables. The base case in this definition is when F has arity zero.

Definition of constraints

$$T ::= \quad X \mid F\,\vec{T}$$

$$\begin{aligned} C ::= \quad & T = T \mid C \wedge C \mid \exists X.C \\ & \mid x \preceq T \\ & \mid \varsigma \preceq T \\ & \mid def\ x = \varsigma\ in\ C \end{aligned}$$

$$\varsigma ::= \quad \forall \bar{X}[C].T$$

$$def\ x : \varsigma\ in\ C \equiv [x \mapsto \varsigma]\ in\ C$$

Constraint interpretation

$$\frac{\psi x \ni \phi T}{\phi, \psi \vdash x \preceq T} \qquad \frac{\binom{\phi}{\psi}\varsigma \ni \phi T}{\phi, \psi \vdash \varsigma \preceq T} \qquad \frac{\phi, \psi\, [x \mapsto \binom{\phi}{\psi}\varsigma] \vdash C}{\phi, \psi \vdash def\ x : \varsigma\ in\ C}$$

- $x \preceq T$ and $\varsigma \preceq T$ are instantiation constraints interpreted as set membership.
- A type variable X denotes a ground type.
- A variable x denotes a set of ground types.
- A valuation ϕ maps a type variables to a ground type.
- A valuation ψ maps a term variable to a set of ground types.

Constraint Generation

$$
\begin{array}{lcl}
[\![x : T]\!] & = & x \preceq T \\
[\![\lambda x.t : T]\!] & = & \exists X_1 X_2 (def\ x : X_1\ in\ [\![t : X_2]\!] \wedge X_1 \rightarrow X_2 = T \\
 & & if X_1, X_2 \notin t, T \\
[\![t_1\ t_2]\!] & = & \exists X.([\![t_1 : X \rightarrow T]\!] \wedge [\![t_2 : X]\!]) \\
 & & if X \notin t_1, t_2, T \\
[\![let\ x = t_1\ in\ t_2 : T]\!] & = & let\ x : (\!|t_1|\!)\ in\ [\![t_2 : T]\!] \\
(\!|t|\!) & = & \forall[[\![t : X]\!]].X
\end{array}
$$

Because constraint generation is now a mapping of an expression t and a type T to a constraint $[\![x : T]\!]$. There is no need for the parameter Γ

Example

let t stand for the term $\lambda x.x : T$, such that the typing of t is correct.

Constraint generation:

$$\begin{array}{lcl} [\![\lambda x.x : T]\!] & = & \exists X_1 X_2 (def\ x : X_1\ in\ [\![x : X_2]\!] \wedge X_1 \rightarrow X_2 = T) \\ [\![\lambda x.x : T]\!] & = & \exists X_1 X_2 (def\ x : X_1\ in\ x \preceq X_2 \wedge X_1 \rightarrow X_2 = T) \end{array}$$

Constraint solving:

$$\begin{array}{lcl} [\![\lambda x.x : T]\!] & = & \exists X_1 X_2 (def\ x : X_1\ in\ x \preceq X_2 \wedge X_1 \rightarrow X_2 = T) \\ [\![\lambda x.x : T]\!] & = & \exists X_1 X_2 (X_1 \preceq X_2 \wedge X_1 \rightarrow X_2 = T) \\ [\![\lambda x.x : T]\!] & = & \exists X_1 X_2 (X_1 = X_2 \wedge X_1 \rightarrow X_2 = T) \\ [\![\lambda x.x : T]\!] & = & \exists X_1 X_2 (X_1 \rightarrow X_1 = T) \\ [\![\lambda x.x : T]\!] & = & (X_1 \rightarrow X_1 = T) \\ & & if\ X_1 \notin t, T \end{array}$$

Jay, B. (2009). *Pattern calculus: computing with functions and structures*, Springer Verlag.

Pottier, F. and Rémy, D. (2005). The essence of ML type inference.

Wells, J. (1999). Typability and type checking in System F are equivalent and undecidable* 1, *Annals of Pure and Applied Logic* **98**(1-3): 111–156.

Wikipedia (2010). Constraint (mathematics) — wikipedia, the free encyclopedia. [Online; accessed 19-October-2010].
URL: *http://en.wikipedia.org/w/index.php?title=Constraint_(mathematics)&oldid=382532950*